Red Hat Virtualization 4.4

Ruby SDK 指南

使用 Red Hat Virtualization Ruby SDK

Last Updated: 2023-09-08

Red Hat Virtualization 4.4 Ruby SDK 指南

使用 Red Hat Virtualization Ruby SDK

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

本指南论述了如何安装和使用 Red Hat Virtualization Ruby 软件开发套件。

目录

第 1 章 概述

注意

Ruby 软件开发套件(SDK)已弃用。以后的发行版本会删除对 Ruby SDK 的支持。

Ruby 软件开发套件是一个 Ruby gem，它可让您与 Ruby 项目中的 Red Hat Virtualization Manager 进行交互。通过下载这些类并将它们添加到您的项目中，您可以访问一系列功能，以实现高级管理任务的自动化。

1.1. 先决条件

要安装 Ruby 软件开发工具包，您必须有：

- 安装了 Red Hat Enterprise Linux 8 的系统。支持服务器和 Workstation 变体。
- Red Hat Virtualization 权利订阅。

1.2. 安装 RUBY SOFTWARE DEVELOPMENT KIT

1. 启用所需的软件仓库：

```
# subscription-manager repos \
    --enable=rhel-8-for-x86_64-baseos-rpms \
    --enable=rhel-8-for-x86_64-appstream-rpms \
    --enable=rhv-4.4-manager-for-rhel-8-x86_64-rpms
```

2. 安装 Ruby Software Development Kit：

```
# dnf install rubygem-ovirt-engine-sdk4
```

或者，您可以使用 **gem** 进行安装：

```
# gem install ovirt-engine-sdk
```

1.3. 依赖项

Ruby Software Development Kit 具有以下依赖项，如果正在使用 **gem**，则必须手动安装该依赖关系：

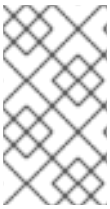
- 用于解析和渲染 XML 的 **libxml2**
- **libcurl** 用于 HTTP 传输
- c 编译器
- 所需的标头和库文件

注意

如果安装了 RPM，则不需要安装依赖文件。

安装依赖项文件：

```
# dnf install gcc libcurl-devel libxml2-devel ruby-devel
```



注意

如果使用 Debian 或 Ubuntu，请使用 **apt-get** ：

```
# apt-get install gcc libxml2-dev libcurl-dev ruby-dev
```

第 2 章 使用软件开发套件

本章定义了 Ruby Software Development Kit 的模块和类，并描述了它们的用法。

2.1. CLASS

OvirtSDK4 模块包含以下软件开发套件类：

连接

Connection 类是连接到服务器并获取对服务树根的引用的机制。有关详细信息，请参阅连接到服务器。

类型

Type 类实施 API 支持的类型。例如，Vm 类是虚拟机的实施。类是数据容器，不包含任何逻辑。您将处理类型的实例。
这些类的实例用作服务方法的参数和返回值。软件开发套件对从底层表示的转换或从中处理。

服务

服务类实施 API 支持的服务。例如，VmsService 类是该服务的实现，用于管理系统中虚拟机集合。
当服务被引用时，SDK 会自动创建这些类的实例。例如，当调用 **SystemService** 类的 **vms_service** 方法时，SDK 会自动创建 **VmsService** 类的新实例：

```
vms_service = connection.system_service.vms_service
```

警告

不要手动创建这些类的实例。构造器参数和方法可能会在以后有所变化。

Error

Error 类是软件开发工具包在报告错误时引发的基本异常类。
某些特定错误类扩展基本错误类：

- AuthError - Authentication or authorization failure
- ConnectionError - 服务器名称无法解析，或者服务器无法访问
- NotFoundError - Requested 对象不存在
- TimeoutError - 操作超时

其他类

其他类（例如，HTTP 客户端类、读者和作者）用于 HTTP 通信和 XML 解析和渲染。不建议使用这些类，因为它们组成了将来可能会改变的内部实施详情。其向后兼容性无法依赖。

2.2. 类型

2.2.1. 创建和修改类型实例

创建或修改类型的实例对服务器端没有任何影响，除非更改被明确发送到服务器调用服务方法，如下所述。服务器端的更改不会自动反映在内存中已存在的实例。

这些类的构造器具有多个可选参数，各自对应于 type 的各个属性。这简化了对象的创建，使用嵌套的调用来处理多个构造器。

在以下示例中，将创建虚拟机的实例及其属性（cluster、template 和 memory）：

使用属性创建虚拟机实例

```
vm = OvirtSDK4::Vm.new(
  name: 'myvm',
  cluster: OvirtSDK4::Cluster.new(
    name: 'mycluster'
  ),
  template: OvirtSDK4::Template.new(
    name: 'mytemplate'
  ),
  memory: 1073741824
)
```

传递给这些构造器的哈希（例如 **：OvirtSDK4::Cluster.new**）将被递归处理。

在以下示例中，使用普通哈希而不是显式为 Cluster 和 Template 类调用构造器。SDK 在内部将哈希值转换为所需的类。

创建机器实例，带有显示为 Plain Hashs 的属性

```
vm = OvirtSDK4::Vm.new(
  name: 'myvm',
  cluster: {
    name: 'mycluster'
  },
  template: {
    name: 'mytemplate'
  },
  memory: 1073741824
)
```

建议以这种方式使用构造器，但不强制要求。

在以下示例中，会创建一个虚拟机实例，它调用中没有参数到构造者。您可以使用 setter 或使用 setter 和 constructors 的组合来逐一添加虚拟机实例的属性。

独立创建虚拟机实例和添加属性

```
vm = OvirtSDK4::Vm.new
vm.name = 'myvm'
vm.cluster = OvirtSDK4::Cluster.new(name: 'mycluster')
vm.template = OvirtSDK4::Template.new(name: 'mytemplate')
vm.memory = 1073741824
```

在 API 规格中定义为对象列表的属性是 Ruby 数组的实现。例如，**Vm** 类型的 **custom_properties** 属性定义为 **CustomProperty** 类型的对象列表。

添加属性列表作为数组

```
vm = OvirtSDK4::Vm.new(
  name: 'myvm',
  custom_properties: [
    OvirtSDK4::CustomProperty.new(...),
    OvirtSDK4::CustomProperty.new(...),
    ...
  ]
)
```

API 中以枚举值定义的属性作为模块中的常数实施，其名称与枚举类型相同。

以下示例演示了如何使用 **VmStatus** enumerated 值来定义 **Vm** 类型的 status 属性。

```
case vm.status
when OvirtSDK4::VmStatus::DOWN
  ...
when OvirtSDK4::VmStatus::IMAGE_LOCKED
  ...
end
```

重要

在 API 规格中，枚举类型的值为小写，因为这是 XML 和 JSON 的惯例。**但是，在 Ruby 中，这些常数是 使用大写的。**

2.2.2. 检索实例属性

您可以使用对应的属性 readers 检索实例属性。

以下示例检索虚拟机实例的名称和内存：

检索虚拟机实例属性

```
puts "vm.name: #{vm.name}"
puts "vm.memory: #{vm.memory}"
vm.custom_properties.each do |custom_property|
  ...
end
```

将实例属性作为链接检索

某些实例属性作为链接返回，需要 后续链接 方法检索数据。在以下示例中，对虚拟机属性的请求进行响应格式化为 XML 并带有链接：

将虚拟机属性作为链接检索

```
<vm id="123" href="/ovirt-engine/api/vms/123">
  <name>myvm</name>
  <link rel="diskattachments" href="/ovirt-engine/api/vms/123/diskattachments/">
  ...
</vm>
```

链接 vm.disk_attachments 不包含实际的磁盘附加。要检索数据， Connection 类提供了一个 后续_link 方法，它使用 href XML 属性的值来检索实际数据。

在以下示例中，按以下链接 允许您进入磁盘附加，然后进入每个磁盘以检索 别名 ：

检索虚拟机服务

```
vm = vm_service.get
```

使用 follow_link 用于 Retrieve Disk Attachment 和 Disk Alias

```
attachments = connection.follow_link(vm.disk_attachments)
attachments.each do |attachment|
  disk = connection.follow_link(attachment.disk)
```

```
  puts "disk.alias: #{disk.alias}"
end
```

2.3. 服务

2.3.1. 检索服务

API 提供了一组服务，每个服务都与服务器路径相关联。例如，管理系统中虚拟机集合的服务位于 /vms 中，而管理具有标识符 123 的虚拟机的服务则位于 /vms/123 中。

在 Ruby 软件开发套件中，服务的树根由系统 服务实施。它通过调用连接的 system_service 方法来获取：

检索系统服务

```
system_service = connection.system_service
```

旦您对系统服务的引用，就可以使用 *_ service 方法（称为服务 locator s）检索对其他服务的引用。

例如，要检索对系统中虚拟机集合的服务的引用，您可以使用 vms_service 服务 locator：

检索其他服务

```
vms_service = system_service.vms_service
```

要检索对使用标识符 123 管理虚拟机的服务的引用，请使用 vm_service 服务的 service locator。服务 locator 使用虚拟机标识符作为参数：

使用标识符检索虚拟机服务

```
vm_service = vms_service.vms_service('123')
```

重要

服务 locator 调用返回的对象是纯服务，不包含数据。例如，在上一示例中检索到的 vm_service Ruby 对象并不是虚拟机的表示。它是用于检索、更新、删除、启动和停止虚拟机的服务。

2.3.2. 服务方法

找到您想要的服务后，您可以调用其服务方法。这些方法向服务器发送请求并执行实际工作。

管理对象集合的服务通常具有 列表和 添加 方法。

管理单个对象的服务通常具有 获取、更新和删除 方法。

服务可能具有额外的操作方法，方法可在检索、创建、更新或删除之外执行操作。这些方法最常用于管理单个对象的服务。

2.3.2.1. Get

get 方法检索单个对象的表示。

以下示例查找并检索带有标识符 123 的虚拟机的表示：

```
# Find the service that manages the virtual machine:
vms_service = system_service.vms_service
vm_service = vms_service.vm_service('123')

# Retrieve the representation of the virtual machine:
vm = vm_service.get
```

其结果将是对应类型的实例。在本例中，结果是 Ruby 类 Vm 的实例。

某些服务的 get 方法支持额外的参数，它们控制了如何检索对象的表示，或者如果有多个参数来检索的表示。

例如，您可能需要在启动后检索虚拟机的将来状态。管理虚拟机的服务的 get 方法支持 next_run 布尔值参数：

检索虚拟机的 next_run 状态

```
# Retrieve the representation of the virtual machine; not the
# current one, but the one that will be used after the next
# boot:
vm = vm_service.get(next_run: true)
```

详情请参阅软件开发套件的 参考文档。

如果无法检索对象，软件开发套件将引发错误 异常，其中包含失败的详细信息。如果您试图检索不存在的对象，会出现这种情况。

注意

调用 service locator 方法永远不会失败，即使对象不存在，因为服务 locator 方法不会向服务器发送请求。

在以下示例中，service locator 方法可以成功，而 get 方法会产生异常：

找到不存在的虚拟机的服务：no Error

```
# Find the service that manages a virtual machine that does
# not exist. This will succeed.
vm_service = vms_service.vm_service('non_existent_VM')
```

检索不存在的虚拟机服务：错误

```
# Retrieve the virtual machine. This will raise an exception.
vm = vm_service.get
```

2.3.2.2. list

列表 方法检索集合中多个对象的表示。

列出虚拟机集合

```
# Find the service that manages the collection of virtual
# machines:
vms_service = system_service.vms_service
vms = vms_service.list
```

结果是一个 Ruby 数组，其中包含相应类型的实例。在上例中，响应是 Ruby 类 Vm 的实例列表。

某些服务的列表 方法支持额外的参数。

例如，几乎所有顶级集合都支持 search 参数，用于过滤结果，而 max 参数用于限制服务器返回的结果数。

列出调用"my*"的虚拟机

```
vms = vms_service.list(search: 'name=my*', max: 10)
```

注意

不是所有列表方法都支持 search 或 max 参数。些列表方法可能支持其他参数。详情请查看 参考文档。

如果结果列表为空，则返回的值是空的 Ruby 数组。它永远不会为 nil。

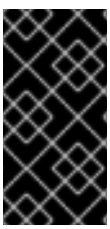
如果无法检索结果列表，则 SDK 会产生一个包含失败详情的 Error 异常。

2.3.2.3. 添加

添加 方法向集合添加新元素。它们收到描述要添加的对象的相关类型实例，发送请求来添加它，以及返回描述添加对象的类型实例。

添加新虚拟机

```
# Add the virtual machine:
vm = vms_service.add(
  OvirtSDK4::Vm.new(
    name: 'myvm',
    cluster: {
      name: 'mycluster'
    },
    template: {
      name: 'mytemplate'
    }
  )
)
```



重要

add 方法返回的 Ruby 对象是相关类型的实例。它并不是一个服务，只是一个数据容器。在上例中，返回的对象是 Vm 类的实例。

如果您需要对刚添加的虚拟机执行操作，您必须找到管理该服务并调用服务 locator：

启动新的虚拟机

```
# Add the virtual machine:
vm = vms_service.add(
  ...
)

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Start the virtual machine:
vm_service.start
```

大多数对象的创建都是异步任务。例如，如果您创建新虚拟机，则添加 方法将在虚拟机完全创建并准备好使用前返回虚拟机。您应该轮询对象的状态，直到对象完全创建。对于表示处于 DOWN 状态之前检查的虚拟机。

推荐的方法是创建虚拟机，找到管理新虚拟机的服务，并重复检索到虚拟机状态为 DOWN，这表示已创建所有磁盘。

添加虚拟机、循环服务并检索到它的状态

```
# Add the virtual machine:
vm = vms_service.add(
  ...
)

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Wait until the virtual machine is DOWN, indicating that all the
# disks have been created:
loop do
  sleep(5)
  vm = vm_service.get
  break if vm.status == OvirtSDK4::VmStatus::DOWN
end
```

如果无法创建对象，则 SDK 将生成包含失败详情的 Error 异常。它永远不会返回 nil。

2.3.2.4. Update（更新）

更新 方法更新现有对象。它们收到描述要执行更新的相关类型的实例，发送请求更新，再返回描述更新对象的类型实例。

注意

这个更新方法返回的 Ruby 对象是相关类型的实例。它并不是一个服务，只是一个数据容器。在这个特定示例中，返回的对象将是 Vm 类的实例。

在以下示例中，service locator 方法找到管理虚拟机的服务，更新 方法更新其名称：

更新虚拟机名称

```
# Find the virtual machine and the service that
# manages it:
vm = vms_service.list(search: 'name=myvm').first
vm_service = vms_service.vm_service(vm.id)

# Update the name:
updated_vm = vms_service.update(
  OvirtSDK4::Vm.new(
    name: 'newvm'
  )
)
```

当您更新对象时，只更新您要更新的属性：

更新虚拟机选项的属性（推荐）

```
vm = vm_service.get
vm.name = 'newvm'
```

不要更新整个对象：

更新虚拟机的所有属性（不推荐）

```
# Retrieve the current representation:
vms_service.update(vm)
```

更新虚拟机的所有属性都浪费资源，并可能会在服务器端引入意外错误。

更新 某些服务的方法支持额外的参数，它们可用于控制如何更新或什么。例如，您可能希望更新虚拟机的内存，而不是处于当前状态，但在下次启动时。管理虚拟机的服务的更新 方法支持 next_run 布尔值参数：

在下一次运行中更新虚拟机的内存

```
vm = vm_service.update(
  OvirtSDK4::Vm.new(
    memory: 1073741824
  ),
  next_run: true
)
```

如果无法进行更新，则 SDK 将生成包含失败详情的 Error 异常。它永远不会返回 nil。

2.3.2.5. 删除

删除 方法删除现有对象。它们通常不支持参数，因为它们是管理单个对象的服务方法，而服务已知道要删除的对象。

使用标识符 123删除虚拟机

```
vm_service = vms_service.vm_service('123')
vms_service.remove
```

某些 删除 方法支持参数控制如何删除参数。例如，可以在保留其磁盘的同时删除虚拟机，使用 detach_only 布尔值参数：

在预保留磁盘时删除虚拟机

```
vm_service.remove(detach_only: true)
```

如果对象被删除，则 remove 方法返回 nil。它不会返回移除的对象。

如果无法删除对象，则 SDK 会出现包含失败详情的 Error 异常。

2.3.2.6. 额外操作

除了上述方法外，还有其他操作方法。管理虚拟机的服务具有启动和停止它的方法。

启动虚拟机

```
vm_service.start
```

某些操作方法包括修改操作的参数。例如，start 方法支持 use_cloud_init 参数。

使用 Cloud-Init 启动虚拟机

```
vm_service.start(use_cloud_init: true)
```

大多数操作方法都会在成功时返回 nil，并在它们失败时引发错误。http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/Error但是，一些操作方法返回值。例如，管理存储域的服务具有 is_attached 操作方法，用于检查存储域是否已附加到数据中心。is_attached action 方法返回一个布尔值：

检查附加存储域

```
sds_service = system_service.storage_domains_service
sd_service = sds_service.storage_domain_service('123')
if sd_service.is_attached
  ...
end
```

请参阅软件开发套件的 参考文档，以查看每个服务、它们参数和返回值支持的操作方法。

第 3 章 RUBY 示例

3.1. 连接到 RED HAT VIRTUALIZATION MANAGER

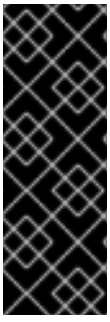
Connection 类是软件开发工具包的入口点。它提供对 Red Hat Virtualization Manager 的 REST API 的服务的访问权限。

Connection 类的参数有：

- URL - Red Hat Virtualization Manager API 的基本 URL
- username
- password
- ca_file - 包含可信 CA 证书的 PEM 文件。当连接到由 TLS 保护的服务器时，需要 ca.pem 文件。如果没有指定 ca_file，则使用系统范围的 CA 证书存储。

连接到 Red Hat Virtualization Manager

```
connection = OvirtSDK4::Connection.new(
  url: 'https://engine.example.com/ovirt-engine/api',
  username: 'admin@internal',
  password: '...',
  ca_file: 'ca.pem',
)
```



重要

连接保存关键资源，包括与服务器的 HTTP 连接和身份验证令牌等。当这些资源不再使用时，您必须释放这些资源：

```
connection.close
```

连接以及从中获取的所有服务在连接关闭后无法使用。

如果连接失败，软件开发套件将引发错误 异常，包含失败的详细信息。

如需更多信息，请参阅 Connection:initialize。

3.2. 列出数据中心

这个 Ruby 示例列出了数据中心。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of data centers:
dcs_service = system_service.data_centers_service

# Retrieve the list of data centers and for each one
# print its name:
dcs = dcs_service.list
dcs.each do |dc|
  puts dc.name
end
```

在只有 Default 数据中心的环境中，示例输出：

```
Default
```

如需更多信息，请参阅 http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/DataCentersService:list。

3.3. 列出集群

这个 Ruby 示例列出了集群。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of clusters:
```

```
cls_service = system_service.clusters_service

# Retrieve the list of clusters and for each one
# print its name:
cls = cls_service.list
cls.each do |cl|
  puts cl.name
end
```

在只有 Default 集群时，示例输出：

```
Default
```

如需更多信息，请参阅 ClustersService:list。

3.4. 列出逻辑网络

这个 Ruby 示例列出了逻辑网络。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of networks:
nws_service = system_service.networks_service

# Retrieve the list of clusters and for each one
# print its name:
nws = nws_service.list
nws.each do |nw|
  puts nw.name
end
```

在一个只使用默认管理网络的环境中，示例输出：

```
ovirtmgmt
```

如需更多信息，请参阅 NetworksService:list。

3.5. 列出主机

此 Ruby 示例列出了主机。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the service that manages the
# collection of hosts:
host_service = system_service.hosts_service

# Retrieve the list of hosts and for each one
# print its name:
host = host_service.list
host.each do |host|
  puts host.name
end
```

在一个环境中，只有一个附加主机(Atlantic)示例输出：

```
Atlantic
```

有关更多信息，请参阅 HostsService:list-instance_method。

3.6. 列出 ISO 存储域中的 ISO 文件

这个 Ruby 示例列出了 ISO 存储域 myiso 中的 ISO 文件。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Find the service that manages the collection of storage domains:
sds_service = system_service.storage_domains_service

# Find the ISO storage domain:
sd = sds_service.list(search: 'name=myiso').first

# Find the service that manages the ISO storage domain:
sd_service = sds_service.storage_domain_service(sd.id)

# Find the service that manages the collection of files available in the storage domain:
files_service = sd_service.files_service

# List the names of the files. Note that the name of the .iso file is contained
# in the `id` attribute.
files = files_service.list
```

```
files.each do |file|
  puts file.id
end
```

如需更多信息，请参阅 FilesService:list。

如果您不知道 ISO 存储域的名称，这个 Ruby 示例会检索所有 ISO 存储域。

```
# Find the ISO storage domain:
iso_sds = sds_service.list.select { |sd| sd.type == OvirtSDK4::StorageDomainType::ISO }
```

如需更多信息，请参阅 StorageDomainsService:list。

3.7. 创建 NFS 存储域

这个 Ruby 示例添加了一个 NFS 存储域。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Get the reference to the storage domains service:
sds_service = connection.system_service.storage_domains_service

# Create a new NFS data storage domain:
sd = sds_service.add(
  OvirtSDK4::StorageDomain.new(
    name: 'mydata',
    description: 'My data',
    type: OvirtSDK4::StorageDomainType::DATA,
    host: {
      name: 'myhost'
    },
    storage: {
      type: OvirtSDK4::StorageType::NFS,
      address: 'server0.example.com',
      path: '/nfs/ovirt/40/mydata'
    }
  )
)

# Wait until the storage domain is unattached:
sd_service = sds_service.storage_domain_service(sd.id)
loop do
  sleep(5)
  sd = sd_service.get
  break if sd.status == OvirtSDK4::StorageDomainStatus::UNATTACHED
end
```

-

如需更多信息，请参阅 http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/StorageDomainsService:add。

3.8. 将存储域附加到数据中心

这个 Ruby 示例将现有的 NFS 存储域 mydata 附加到现有数据中心 mydc。这个示例用于附加数据和 ISO 存储域。

```
# Get the reference to the root of the services tree:
system_service = connection.system_service

# Locate the service that manages the storage domains and use it to
# search for the storage domain:
sds_service = system_service.storage_domains_service
sd = sds_service.list(search: 'name=mydata')[0]

# Locate the service that manages the data centers and use it to
# search for the data center:
dcs_service = system_service.data_centers_service
dc = dcs_service.list(search: 'name=mydc')[0]

# Locate the service that manages the data center where you want to
# attach the storage domain:
dc_service = dcs_service.data_center_service(dc.id)

# Locate the service that manages the storage domains that are attached
# to the data centers:
attached_sds_service = dc_service.storage_domains_service

# Use the "add" method of service that manages the attached storage
# domains to attach it:
attached_sds_service.add(
  OvirtSDK4::StorageDomain.new(
    id: sd.id
  )
)

# Wait until the storage domain is active:
attached_sd_service = attached_sds_service.storage_domain_service(sd.id)
loop do
  sleep(5)
  sd = attached_sd_service.get
  break if sd.status == OvirtSDK4::StorageDomainStatus::ACTIVE
end
```

如需更多信息，请参阅 http://www.rubydoc.info/gems/ovirt-engine-sdk/OvirtSDK4/StorageDomainsService:add。

3.9. 创建虚拟机

这个 Ruby 示例创建一个虚拟机。这个示例使用带有符号和嵌套哈希的哈希值作为其值。另一种方法是直接使用对应对象的构造者，更加详细。如需更多信息，请参阅使用属性 创建虚拟机实例。

```
# Get the reference to the "vms" service:
vms_service = connection.system_service.vms_service

# Use the "add" method to create a new virtual machine:
vms_service.add(
  OvirtSDK4::Vm.new(
    name: 'myvm',
    cluster: {
      name: 'mycluster'
    },
    template: {
      name: 'Blank'
    }
  )
)
```

创建虚拟机后，建议 轮询虚拟机的状态，以确保已创建所有磁盘。

如需更多信息，请参阅 VmsService:add。

3.10. 创建 VNIC 配置集

这个 Ruby 示例创建了一个 vNIC 配置集。

```
# Find the root of the tree of services:
system_service = connection.system_service

# Find the network where you want to add the profile. There may be multiple
# networks with the same name (in different data centers, for example).
# Therefore, you must look up a specific network by name, in a specific data center.
dcs_service = system_service.data_centers_service
dc = dcs_service.list(search: 'name=mydc').first
networks = connection.follow_link(dc.networks)
network = networks.detect { |n| n.name == 'mynetwork' }

# Create the vNIC profile, with passthrough and port mirroring disabled:
profiles_service = system_service.vnic_profiles_service
profiles_service.add(
  OvirtSDK4::VnicProfile.new(
    name: 'myprofile',
    pass_through: {
```

```
      mode: OvirtSDK4::VnicPassThroughMode::DISABLED,
    },
    port_mirroring: false,
    network: {
      id: network.id
    }
  )
)
```

如需更多信息，请参阅 VnicProfilesService:add。

3.11. 创建 VNIC

要确保新创建的虚拟机具有网络访问，您必须创建并附加 vNIC。

此 Ruby 示例创建一个 vNIC，并将其附加到现有虚拟机 myvm 中。

```
# Find the root of the tree of services:
system_service = connection.system_service

# Find the virtual machine:
vms_service = system_service.vms_service
vm = vms_service.list(search: 'name=myvm').first

# In order to specify the network that the new NIC will be connected to, you must
# specify the identifier of the vNIC profile. However, there may be multiple
# profiles with the same name (for different data centers, for example), so first
# you must find the networks that are available in the cluster that the
# virtual machine belongs to.
cluster = connection.follow_link(vm.cluster)
networks = connection.follow_link(cluster.networks)
network_ids = networks.map(&:id)

# Now that you know what networks are available in the cluster, you can select a
# vNIC profile that corresponds to one of those networks, and has the
# name that you want to use. The system automatically creates a vNIC
# profile for each network, with the same name as the network.
profiles_service = system_service.vnic_profiles_service
profiles = profiles_service.list
profile = profiles.detect { |p| network_ids.include?(p.network.id) && p.name == 'myprofile' }

# Locate the service that manages the network interface cards collection of the
# virtual machine:
nics_service = vms_service.vm_service(vm.id).nics_service

# Add the new network interface card:
nics_service.add(
  OvirtSDK4::Nic.new(
    name: 'mynic',
```

```
    description: 'My network interface card',
    vnic_profile: {
      id: profile.id
    }
  )
)
```

如需更多信息，请参阅 VmsService:add。

3.12. 创建虚拟磁盘

要确保新创建的虚拟机可以访问持久性存储，您必须创建并附加磁盘。

这个 Ruby 示例创建并附加虚拟存储磁盘。

```
# Locate the virtual machines service and use it to find the virtual
# machine:
vms_service = connection.system_service.vms_service
vm = vms_service.list(search: 'name=myvm')[0]

# Locate the service that manages the disk attachments of the virtual
# machine:
disk_attachments_service = vms_service.vm_service(vm.id).disk_attachments_service

# Use the "add" method of the disk attachments service to add the disk.
# Note that the size of the disk, the `provisioned_size` attribute, is
# specified in bytes, so to create a disk of 10 GiB the value should
# be 10 * 2^30.
disk_attachment = disk_attachments_service.add(
  OvirtSDK4::DiskAttachment.new(
    disk: {
      name: 'mydisk',
      description: 'My disk',
      format: OvirtSDK4::DiskFormat::COW,
      provisioned_size: 10 * 2**30,
      storage_domains: [{
        name: 'mydata'
      }]
    },
    interface: OvirtSDK4::DiskInterface::VIRTIO,
    bootable: false,
    active: true
  )
)

# Wait until the disk status is OK:
disks_service = connection.system_service.disks_service
disk_service = disks_service.disk_service(disk_attachment.disk.id)
loop do
```

```
  sleep(5)
  disk = disk_service.get
  break if disk.status == OvirtSDK4::DiskStatus::OK
end
```

如需更多信息，请参阅 DiskAttachmentsService:add。

3.13. 将 ISO 镜像附加到虚拟机

这个 Ruby 示例将 CD-ROM 附加到虚拟机，并将其改为 ISO 镜像以安装客户端操作系统。

```
# Get the reference to the "vms" service:
vms_service = connection.system_service.vms_service

# Find the virtual machine:
vm = vms_service.list(search: 'name=myvm')[0]

# Locate the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Locate the service that manages the CDROM devices of the VM:
cdroms_service = vm_service.cdroms_service

# List the first CDROM device:
cdrom = cdroms_service.list[0]

# Locate the service that manages the CDROM device you just found:
cdrom_service = cdroms_service.cdrom_service(cdrom.id)

# Change the CD of the VM to 'my_iso_file.iso'. By default this
# operation permanently changes the disk that is visible to the
# virtual machine after the next boot, but it does not have any effect
# on the currently running virtual machine. If you want to change the
# disk that is visible to the current running virtual machine, change
# the `current` parameter's value to `true`.
cdrom_service.update(
  OvirtSDK4::Cdrom.new(
    file: {
      id: 'CentOS-7-x86_64-DVD-1511.iso'
    }
  ),
  current: false
)
```

如需更多信息，请参阅 VmService:cdroms_service。

3.14. 分离虚拟磁盘

这个 Ruby 示例从虚拟机中分离磁盘。

```
# Find the virtual machine:
vm = vms_service.list(search: 'name=myvm').first

# Detach the first disk from the virtual machine:
vm_service = vms_service.vm_service(vm.id)
attachments_service = vm_service.disk_attachments_service
attachment = attachments_service.list.first

# Remove the attachment. The default behavior is that the disk is detached
# from the virtual machine, but not deleted from the system. If you wish to
# delete the disk, change the `detach_only` parameter to `false`.
attachment.remove(detach_only: true)
```

如需更多信息，请参阅 VmService:disk_attachments_service。

3.15. 启动虚拟机

这个 Ruby 示例启动虚拟机。

```
# Get the reference to the "vms" service:
vms_service = connection.system_service.vms_service

# Find the virtual machine:
vm = vms_service.list(search: 'name=myvm')[0]

# Locate the service that manages the virtual machine, as that is where
# the action methods are defined:
vm_service = vms_service.vm_service(vm.id)

# Call the "start" method of the service to start it:
vm_service.start

# Wait until the virtual machine status is UP:
loop do
  sleep(5)
  vm = vm_service.get
  break if vm.status == OvirtSDK4::VmStatus::UP
end
```

如需更多信息，请参阅 VmService:start。

3.16. 启动具有特定引导设备和引导顺序的虚拟机

这个 Ruby 示例启动指定引导设备和引导顺序的虚拟机。

```
# Find the root of the tree of services:
system_service = connection.system_service

# Find the virtual machine:
vms_service = system_service.vms_service
vm = vms_service.list(search: 'name=myvm').first

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Start the virtual machine explicitly indicating the boot devices and order:
vm_service.start(
  vm: {
    os: {
      boot: {
        devices: [
          OvirtSDK4::BootDevice::NETWORK,
          OvirtSDK4::BootDevice::CDROM
        ]
      }
    }
  }
)

# Wait until the virtual machine is up:
loop do
  sleep(5)
  vm = vm_service.get
  break if vm.status == OvirtSDK4::VmStatus::UP
end
```

如需更多信息，请参阅 BootDevice。

3.17. 使用 CLOUD-INIT 启动虚拟机

此 Ruby 示例使用 Cloud-Init 工具启动虚拟机，以设置 root 密码和网络配置。

```
# Find the virtual machine:
vms_service = connection.system_service.vms_service
vm = vms_service.list(search: 'name=myvm')[0]

# Find the service that manages the virtual machine:
vm_service = vms_service.vm_service(vm.id)

# Create a cloud-init script to execute in the
# deployed virtual machine. The script must be correctly
# formatted and indented because it uses YAML.
```

```
my_script = ''
write_files:
 - content: |
     Hello, world!
   path: /tmp/greeting.txt
   permissions: '0644'
''

# Start the virtual machine, enabling cloud-init and providing the
# password for the `root` user and the network configuration:
vm_service.start(
 use_cloud_init: true,
 vm: {
  initialization: {
   user_name: 'root',
   root_password: 'redhat123',
   host_name: 'myvm.example.com',
   nic_configurations: [
    {
     name: 'eth0',
     on_boot: true,
     boot_protocol: OvirtSDK4::BootProtocol::STATIC,
     ip: {
      version: OvirtSDK4::IpVersion::V4,
      address: '192.168.0.100',
      netmask: '255.255.255.0',
      gateway: '192.168.0.1'
     }
    }
   ],
   dns_servers: '192.168.0.1 192.168.0.2 192.168.0.3',
   dns_search: 'example.com',
   custom_script: my_script
  }
 }
)
```

如需更多信息，请参阅 VmService:start。

3.18. 检查系统事件

这个 Ruby 示例检索日志记录的系统事件。

```
# In order to ensure that no events are lost, it is recommended to write
# the index of the last processed event, in persistent storage.
# Here, it is stored in a file called `index.txt`. In a production environment,
# it will likely be stored in a database.
INDEX_TXT = 'index.txt'.freeze

def write_index(index)
  File.open(INDEX_TXT, 'w') { |f| f.write(index.to_s) }
```

```
end

def read_index
  return File.read(INDEX_TXT).to_i if File.exist?(INDEX_TXT)
  nil
end

# This is the function that is called to process the events. It prints
# the identifier and description of each event.
def process_event(event)
  puts("#{event.id} - #{event.description}")
end

# Find the root of the tree of services:
system_service = connection.system_service

# Find the service that manages the collection of events:
events_service = system_service.events_service

# If no index is stored yet, retrieve the last event and start with it.
# Events are ordered by index, in ascending order. `max=1` retrieves only one event,
# the last event.
unless read_index
  events = events_service.list(max: 1)
  unless events.empty?
    first = events.first
    process_event(first)
    write_index(first.id.to_i)
  end
end

# This loop retrieves the events, always starting from the last index. It waits
# before repeating. The `from` parameter specifies that you want to retrieve
# events that are newer than the last index that was processed. Note: the `max`
# parameter is not used, so that all pending events will be retrieved.
loop do
  sleep(5)
  events = events_service.list(from: read_index)
  events.each do |event|
    process_event(event)
    write_index(event.id.to_i)
  end
end
```

如需更多信息，请参阅 **EventsService:list**。

附录 A. 法律通知